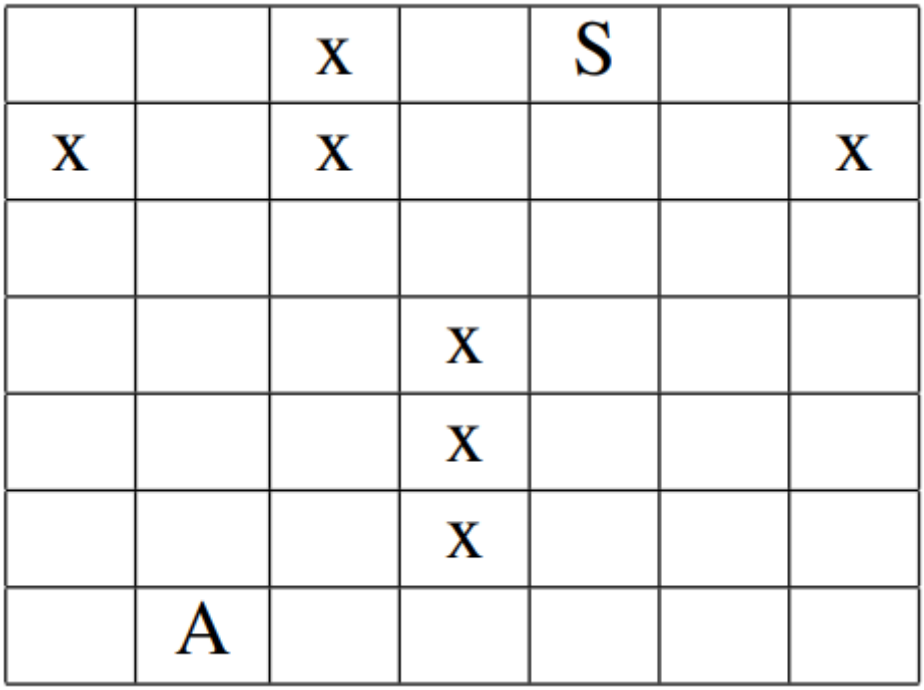
UNIVERSIDADE DE ÉVORA

5º Semestre
Ano Letivo 2020/2021

Inteligência Artificial

1º Trabalho Prático -

Resolução de problemas como problemas de pesquisa no espaço de estados



Trabalho realizado por:

Pedro Grilo, 43012

Diogo Castanho, 42496

Rodrigo Vicente, 42844

Introdução

Com a resolução deste trabalho é pretendido que sejam utilizadas as capacidades de resoluções de problemas de pesquisa no espaço de estados.

Para isso, deveremos conseguir perceber as diferenças entre pesquisas informadas e pesquisas não informadas, como também os diferentes algoritmos em cada uma das duas pesquisas.

Temos o objetivo de conseguir achar o algoritmo mais eficiente (para isso, temos que testar todas as variantes) para resolver os problemas dados.

Resolução dos Exercícios Propostos

1º Exercício

a) Para representar o espaço de dados usámos um tuplo com as coordenas iniciais, e outro com as coordenadas finais:

```
estado_inicial((2,7)).
estado_final((5,1)).
```

Para representar as casas onde o agente não pode passar (casas bloqueadas) utilizámos também tuplos com as coordenadas respetivas:

```
bloqueada((1,2)).
bloqueada((3,1)).
bloqueada((3,2)).

bloqueada((4,4)).
bloqueada((4,5)).
bloqueada((4,6)).

bloqueada((7,2)).
```

Para representar as ações que o agente pode fazer, definimos um predicado op/4, com as seguintes ações: cima, direita, baixo, esquerda.

```
op(e(X,Y),cima,e(X,Y1),1) :-
    Y > 1,
    Y1 is Y-1,
    \+ bloqueada(e(X,Y1)).

op(e(X,Y),direita,e(X1,Y), 1) :-
   X < 7,
   X1 is X+1,
   \+ bloqueada(e(X1,Y)).

op(e(X,Y),baixo,e(X,Y1),1) :-
    Y < 7,
    Y1 is Y+1,
    \+ bloqueada(e(X,Y1)).

op(e(X,Y),esquerda,e(X1,Y),1) :-
    X > 1,
    X1 is X-1,
    \+ bloqueada(e(X1,Y)).
```

- Tuplo (X,Y) representa a posição inicial e o segundo Tuplo (X,Y1, no primeiro caso) a posição onde o agente irá ficar após ir para a operação pretendida.
- Para impedir que o agente vá para uma casa bloqueada, utilizamos o \+ bloqueada, com o argumento da posição para a qual ele vai, só permitindo que **op** seja concluído se essa mesma não for então uma casa bloqueada.

b) De modo a conseguirmos perceber qual o algoritmo de pesquisa mais eficiente, implementámos para a pesquisa informada a pesquisa em profundidade, a pesquisa em largura e a pesquisa em profundidade iterativa, faladas nas aulas e que, a nosso ver, poderiam ser vantajosas para este problema.

Após a implementação e o teste dos três algoritmos, conseguimos perceber que o algoritmo mais eficiente era o da pesquisa em profundidade.

De seguida estarão os resultados obtidos numa tabela.

c) Obtivemos os seguintes resultados para as duas pesquisas implementadas:

Algoritmo de pesquisa\Fator	Estados Visitados	Número máximo de nós em memória	Custo	Profundidade
Profundidade	13 estados	13 nós em memória	9	9
Largura	116 estados	57 nós em memória	9	9
Profundidade iterativa	22449 estados	13 nós em memória	9	9

Com as seguintes imagens dos dados da tabela:

- Para a pesquisa em largura

```
custo(9)
profundidade(9)
e([],e(2,7))
e(cima,e(2,6))
e(cima,e(2,5))
e(cima,e(2,4))
e(cima,e(2,3))
e(direita,e(3,3))
e(direita,e(4,3))
e(cima,e(4,2))
e(cima,e(4,1))
e(direita,e(5,1))
nos(visitados(116),lista(57))
```

- Para a pesquisa em profundidade

```
custo(9)
profundidade(9)
e([],e(2,7))
e(cima,e(2,6))
e(cima,e(2,5))
e(cima,e(2,4))
e(cima,e(2,3))
e(direita,e(3,3))
e(direita,e(4,3))
e(cima,e(4,2))
e(cima,e(4,1))
e(direita,e(5,1))
nos(visitados(13),lista(13))
```

- Para a pesquisa em profundidade iterativa

```
custo(9)
profundidade(9)
e([],e(2,7))
e(cima,e(2,6))
e(cima,e(2,5))
e(cima,e(2,4))
e(cima,e(2,3))
e(direita,e(3,3))
e(direita,e(4,3))
e(cima,e(4,2))
e(cima,e(4,1))
e(direita,e(5,1))
nos(visitados(22449),lista(13))
```

Pelo que, o algoritmo mais eficiente é a pesquisa em profundidade, mostrando uma grande vantagem face aos outros dois algoritmos, sendo que:

- Número de nós visitados: 13
- Número máximo de nós em memória: 13
- Custo: 9
- Profundidade: 9

Instruções para compilar com a pesquisa não informada:

1. Abrir o terminal na pasta com os ficheiros
2. Abrir prolog
3. Compilar [pni2].
4. Fazer pesquisa(agente, *nome da pesquisa*), sendo que nome da pesquisa pode ser: **profundidade**, **largura** ou **it**.

d) As duas heurísticas que pensámos para este problema foram a distância de Manhattan e a distância Euclidiana.

- **Distância de Manhattan**: onde calculamos a diferença entre as coordenadas finais e as coordenadas iniciais, e somamos esse valor
- **Distância Euclidiana**: onde se calcula a raiz quadrada da soma dos quadrados da diferença entre as coordenadas finais e iniciais

e) Para percebemos qual o algoritmo mais eficiente para resolver o problema, implementámos para as pesquisas informadas (e usando as duas heurísticas), as pesquisas A* e Greedy (ou ansiosa).
Os resultados que obtivemos levaram-nos a crer que a pesquisa Greedy, só por si, era melhor nas duas heurísticas que a pesquisa A*.
Comparando as duas heurísticas com o algoritmo Greedy, percebemos que obtemos a melhor eficiência usando a heurística da distância de Manhattan.

f) Obtivemos os seguintes resultados para as duas pesquisas implementadas (com as diferentes heurísticas):

(Sabendo que heurística 1 é distância de Manhattan e heurística 2 é distância Euclidiana)

Algoritmo de pesquisa\Fator	Estados Visitados	Número máximo de nós em memória	Custo	Profundidade
A* com H1	60 estados	43 nós	9	9
A* com H2	66 estados	39 nós	9	9
Greedy com H1	11 estados	14 nós	9	9
Greedy com H2	11 estados	15 nós	9	9

Para a pesquisa **A*** com heurística 1:

```
custo(9)
profundidade(9)
e([],e(2,7))
e(cima,e(2,6))
e(cima,e(2,5))
e(cima,e(2,4))
e(cima,e(2,3))
e(direita,e(3,3))
e(direita,e(4,3))
e(cima,e(4,2))
e(cima,e(4,1))
e(direita,e(5,1))
nos(visitados(60),lista(43))
```

- Nós visitados: 60
- Máximo de nós em memória: 43
- Custo: 9
- Profundidade: 9

Para a pesquisa **A*** com heurística 2:

```
custo(9)
profundidade(9)
e([],e(2,7))
e(cima,e(2,6))
e(cima,e(2,5))
e(cima,e(2,4))
e(cima,e(2,3))
e(direita,e(3,3))
e(direita,e(4,3))
e(cima,e(4,2))
e(cima,e(4,1))
e(direita,e(5,1))
nos(visitados(66),lista(39))
```

- Nós visitados: 66
- Máximo de nós em memória: 39
- Custo: 9
- Profundidade: 9

Para a pesquisa **Greedy** com heurística 1:

```
custo(9)
profundidade(9)
e([],e(2,7))
e(cima,e(2,6))
e(cima,e(2,5))
e(cima,e(2,4))
e(cima,e(2,3))
e(direita,e(3,3))
e(direita,e(4,3))
e(cima,e(4,2))
e(cima,e(4,1))
e(direita,e(5,1))
nos(visitados(11),lista(14))
```

- Nós visitados: 11
- Máximo de nós em memória: 14
- Custo: 9
- Profundidade: 9

Para a pesquisa **Greedy** com heurística 2:

```
custo(9)
profundidade(9)
e([],e(2,7))
e(cima,e(2,6))
e(cima,e(2,5))
e(cima,e(2,4))
e(cima,e(2,3))
e(direita,e(3,3))
e(direita,e(4,3))
e(cima,e(4,2))
e(cima,e(4,1))
e(direita,e(5,1))
nos(visitados(11),lista(15))
```

- Nós visitados: 11
- Máximo de nós em memória: 15
- Custo: 9
- Profundidade: 9

Comparando agora as pesquisas não informadas, com as informadas, conseguimos ver que para este problema em concreto, a pesquisa em profundidade teve um rendimento muito igual às melhores pesquisas informadas (Greedy H1 e H2), provavelmente pelo facto de ser um problema com pouco "espaço" e tamanho, sendo que, de resto, as pesquisas informadas têm uma eficiência muito superior a qualquer umas das outras pesquisas não informadas.

Instruções para compilar com a pesquisa não informada:

1. Abrir o terminal na pasta com os ficheiros
2. Abrir prolog
3. Compilar [pni3].
4. Fazer pesquisa(agente, *nome da pesquisa*), sendo que nome da pesquisa pode ser: **a** ou **g**.
5. *Nota*: para mudar a heurística, comentar no ficheiro pni3.pl uma delas, e deixar a outra

2º Exercício

a) Para representar o espaço de dados usámos um tuplo com as coordenas iniciais, os dois primeiros números, do agente, e os seguintes dois, da caixa.
Para o estado final, é igualmente um tuplo, mas apenas necessitamos de ter em conta o estado final da caixa (5,1), pelo que os "_" representam qualquer posição desde que a final da caixa seja essa mesmo.

```
estado_inicial(p(2,7,2,6)).
estado_final(p(_,_,5,1)).
```

Para representar as casas onde o agente não pode passar (casas bloqueadas) utilizámos também tuplos com as coordenadas respetivas (igual ao primeiro exercício).

```
bloqueada((1,2)).
bloqueada((3,1)).
bloqueada((3,2)).

bloqueada((4,4)).
bloqueada((4,5)).
bloqueada((4,6)).

bloqueada((7,2)).
```

Para representar as ações que o agente pode fazer, definimos um predicado op/4, com as seguintes ações: cima, direita, baixo, esquerda.

Criámos um predicado auxiliar, iguais, que permite ver se os tuplos são ou não iguais, como também um predicado limite, que simboliza os limites do tabuleiro (irá ser usado para prevenir que o agente ou a caixa faça uma movimentação fora do tabuleiro).

Se os tuplos forem iguais, o que quer dizer que a o movimento do agente vai parar a um estado onde está a caixa, então a caixa irá

mover-se igualmente como o agente (sem exceder os limites, ou ir para uma casa bloqueada).

Se os tuplos não forem iguais, apenas o agente se irá mover, tendo também, os cuidados com os limites e as casas bloqueadas.

```
% Operação do agente seguir para cima
op(p(X,Y,P,Q),cima,p(X,Y1,P,Q1),1) :-

    Y1 is Y-1,

    (iguais(p(X,Y1),p(P,Q)) ->
        (
            Q1 is Q-1,
            lim(P,Q1),
            \+ bloqueada(p(P,Q1))
        );
        (
            Q1 is Q,
            lim(X,Y1),
            \+ bloqueada(p(X,Y1))
        )
    ).

% Operação do agente seguir para a direita
op(p(X,Y,P,Q),direita,p(X1,Y,P1,Q),1) :-

    X1 is X+1,

    (iguais(p(X1,Y),p(P,Q)) ->
        (
            P1 is P+1,
            lim(P1,Q),
            lim(X1,Y),
            \+ bloqueada(p(P1,Q))
        );
        (
            P1 is P,
            lim(X1,Y),
            \+ bloqueada(p(X1,Y))
        )
    ).
```

```
% Operação do agente seguir para baixo
op(p(X,Y,P,Q),baixo,p(X,Y1,P,Q1),1) :-

    Y1 is Y+1,

    (iguais(p(X,Y1),p(P,Q)) ->
        (
            Q1 is Q+1,
            lim(P,Q1),
            lim(X,Y1),
            \+ bloqueada(p(P,Q1))
        );
        (
            Q1 is Q,
            lim(X,Y1),
            \+ bloqueada(p(X,Y1))
        )
    ).

% Operação do agente seguir para a esquerda
op(p(X,Y,P,Q),esquerda,p(X1,Y,P1,Q),1) :-

    X1 is X-1,

    (iguais(p(X1,Y),p(P,Q)) ->
        (
            P1 is P-1,
            lim(P1,Q),
            lim(X1,Y),
            \+ bloqueada(p(P1,Q))
        );
        (
            P1 is P,
            lim(X1,Y),
            \+ bloqueada(p(X1,Y))
        )
    ).
```

```
%Limites do tabuleiro
lim(X,Y) :- X=<7, X>=1, Y=<7,Y>=1.
```

g) De modo a conseguirmos perceber qual o algoritmo de pesquisa mais eficiente, implementámos novamente para a pesquisa informada a pesquisa em profundidade, a pesquisa em largura e a pesquisa em profundidade iterativa.

Após a implementação e o teste dos três algoritmos, conseguimos perceber que a pesquisa em profundidade visitou menos estados e guardou menos nós em memória, mas teve um custo e uma profundidade muitos superiores aos da largura. Pelo que, achámos o algoritmo da pesquisa em largura mais eficaz.

O algoritmo da profundidade iterativa, por alguma razão, não conseguia compilar (loop infinito), pelo que não tem resultados na tabela seguinte.

Algoritmo de pesquisa\Fator	Estados Visitados	Número máximo de nós em memória	Custo	Profundidade
Profundidade	187 estados	50 nós em memória	58	58
Largura	3775 estados	582 nós em memória	16	16
Profundidade iterativa	X	X	X	X

h) Obtivemos os seguintes resultados para as duas pesquisas implementadas:

Com as seguintes imagens dos dados da tabela:

- Para a pesquisa em largura

```
custo(16)
profundidade(16)
e([],p(2,7,2,6))
e(cima,p(2,6,2,5))
e(cima,p(2,5,2,4))
e(cima,p(2,4,2,3))
e(esquerda,p(1,4,2,3))
e(cima,p(1,3,2,3))
e(direita,p(2,3,3,3))
e(direita,p(3,3,4,3))
e(direita,p(4,3,5,3))
e(cima,p(4,2,5,3))
e(direita,p(5,2,5,3))
e(direita,p(6,2,5,3))
e(baixo,p(6,3,5,3))
e(baixo,p(6,4,5,3))
e(esquerda,p(5,4,5,3))
e(cima,p(5,3,5,2))
e(cima,p(5,2,5,1))
nos(visitados(3775),lista(582))
```

- Para a pesquisa em profundidade

```
custo(58)
profundidade(58)
```

```
nos(visitados(187),lista(50))
```

Pelo que, o algoritmo mais eficiente é a pesquisa em profundidade, sendo que:

- Número de nós visitados: 187
- Número máximo de nós em memória: 50
- Custo: 58
- Profundidade: 58

Instruções para compilar com a pesquisa não informada:

5. Abrir o terminal na pasta com os ficheiros
6. Abrir prolog
7. Compilar [pni2].
8. Fazer pesquisa(agente2, *nome da pesquisa*), sendo que nome da pesquisa pode ser: **profundidade**, **largura** ou **it** (dá loop infinito).

i) As duas heurísticas que pensámos para este problema foram a distância de Manhattan (como no 1º exercício) e a distância absoluta entre o X final e o X inicial da caixa.

```
%Heuristica 1 - Distancia entre o estado atual e o estado final da caixa
h(p(_,_,Ix,Iy),SOMA):-
    estado_final(p(_,_,Fx,Fy)),
    Dx is abs(Ix - Fx),
    Dy is abs(Iy - Fy),
    SOMA is Dx + Dy.

%Heuristica 2 - Distancia entre o x inicial da caixa e o x final da caixa
h(p(_,_,Ix,_),SOMA):-
    estado_final(p(_,_,Fx,_)),
    Dx is abs(Ix - Fx),
    SOMA is Dx.
```

j) Para percebemos qual o algoritmo mais eficiente para resolver o problema, implementámos para as pesquisas informadas (e usando as duas heurísticas), as pesquisas A* e Greedy (ou ansiosa).

k) Obtivemos os seguintes resultados para as duas pesquisas implementadas (com as diferentes heurísticas):

Algoritmo de pesquisa\Fator	Estados Visitados	Número máximo de nós em memória	Custo	Profundidade
A* com H1	1166 visitados	436 nós	16	16
A* com H2	Stack Overflow			
Greedy com H1	331 estados	51 nós	16	16
Greedy com H2	987 visitados	179 nós	20	20

(Sabendo que heurística 1 é distância de Manhattan e heurística 2 é distância absoluta entre o X final e o X inicial dos estados da caixa)

Para a pesquisa **A*** com heurística 1:

```
custo(9)
profundidade(9)
e([],e(2,7))
e(cima,e(2,6))
e(cima,e(2,5))
e(cima,e(2,4))
e(cima,e(2,3))
e(direita,e(3,3))
e(direita,e(4,3))
e(cima,e(4,2))
e(cima,e(4,1))
e(direita,e(5,1))
nos(visitados(66),lista(39))
```

- Nós visitados: 1166
- Máximo de nós em memória: 436
- Custo: 16
- Profundidade: 16

Para a pesquisa **Greedy** com heurística 1:

```
custo(16)
profundidade(16)
e([],p(2,7,2,6))
e(cima,p(2,6,2,5))
e(cima,p(2,5,2,4))
e(cima,p(2,4,2,3))
e(esquerda,p(1,4,2,3))
e(cima,p(1,3,2,3))
e(direita,p(2,3,3,3))
e(direita,p(3,3,4,3))
e(direita,p(4,3,5,3))
e(cima,p(4,2,5,3))
e(direita,p(5,2,5,3))
e(direita,p(6,2,5,3))
e(baixo,p(6,3,5,3))
e(baixo,p(6,4,5,3))
e(esquerda,p(5,4,5,3))
e(cima,p(5,3,5,2))
e(cima,p(5,2,5,1))
nos(visitados(331),lista(51))
```

- Nós visitados: 331
- Máximo de nós em memória: 51
- Custo: 16
- Profundidade: 16

Para a pesquisa **Greedy** com heurística 2:

```
custo(20)
profundidade(20)
e([],p(2,7,2,6))
e(esquerda,p(1,7,2,6))
e(cima,p(1,6,2,6))
e(direita,p(2,6,3,6))
e(baixo,p(2,7,3,6))
e(direita,p(3,7,3,6))
e(cima,p(3,6,3,5))
e(cima,p(3,5,3,4))
e(cima,p(3,4,3,3))
e(esquerda,p(2,4,3,3))
e(cima,p(2,3,3,3))
e(direita,p(3,3,4,3))
e(direita,p(4,3,5,3))
e(cima,p(4,2,5,3))
e(direita,p(5,2,5,3))
e(direita,p(6,2,5,3))
e(baixo,p(6,3,5,3))
e(baixo,p(6,4,5,3))
e(esquerda,p(5,4,5,3))
e(cima,p(5,3,5,2))
e(cima,p(5,2,5,1))
nos(visitados(987),lista(179))
```

- Nós visitados: 987
- Máximo de nós em memória: 179
- Custo: 20
- Profundidade: 20

Pelo que, é possível concluir que o algoritmo mais eficiente é o da pesquisa Greedy com a heurística da distância de Manhattan. Tem o custo e a profundidade ideal (16), e tem quer menos estados visitados, quer menos nós em memória, comparativamente, a todos os outros algoritmos com heurísticas.

Comparando agora as pesquisas não informadas, com as informadas, conseguimos ver que para este problema em concreto, a melhor pesquisa por muito é a greedy com heurística 1, sendo muito mais eficiente que as pesquisas não informadas (profundidade e largura, apesar da largura ter o mesmo custo e profundidade que esta), mas também melhor que as suas concorrentes com heurísticas.

Para além disso, podemos ver que a pesquisa em largura tem uma eficiência muito próxima da pesquisa A* com h1 (a diferença é que visita e guarda mais nós), mas é melhor que a pesquisa greedy com h2 (tem melhor custo e profundidade).

Instruções para compilar com a pesquisa não informada:

6. Abrir o terminal na pasta com os ficheiros
7. Abrir prolog
8. Compilar [pni3].
9. Fazer pesquisa(agente2, *nome da pesquisa*), sendo que nome da pesquisa pode ser: **a** ou **g**.

Nota: para mudar a heurística, comentar no ficheiro pni3.pl uma delas, e deixar a outra

Conclusão

Com a realização deste trabalho ficámos a ter mais conhecimento sobre a resolução de problemas de pesquisa no espaço de dados, pois aprofundámos e usámos numa vertente mais prática todos os tipos de pesquisa dados nas aulas em problemas concretos.

Passámos pela pesquisa não informada, com os algoritmos de profundidade, largura, e profundidade iterativa, e pela pesquisa informada, com os algoritmos A* e Greedy.

Este contacto prático mostrou-nos como realmente funcionam estas pesquisas, podendo vir a ser muito uteis futuramente.

Pensamos ter conseguido atingir todos os objetivos, apesar de alguns valores nos parecerem um pouco exagerados (por exemplo, no número de nós visitados para certos algoritmos), mas isso pode dever-se ao predicado **op** que nós implementámos (poderá este mesmo não estar o mais eficiente possível).